REMARKS ON THE DEVELOPMENT OF A MULTIBLOCK THREE-DIMENSIONAL EULER CODE FOR OUT OF CORE AND MULTIPROCESSOR CALCULATIONS

by

Antony Jameson, Stefan Leicher
and Jef Dawson

February 1985

1. Introduction

Our purpose in this paper is to describe some of the problems which were presented by the development of a three-dimensional Euler code with a multiblock grid structure in which only a single block at a time is held in core, and the main data base is out of core. The initial development of the code was motivated by the need to use a much denser mesh than could be accomodated in the then available supercomputers, which had one or two million words of memory, in order to provide adequate resolution of complex three-dimensional flows.

The code was based on the explicit multistage time stepping algorithm which had already been demonstrated to yield good accuracy with substantially lower computational costs than previously available algorithms [1,2]. While it was not emphasized in some of the earlier discussions of this scheme, one of the principal objectives which motivated its design had in fact been the need to find an algorithm which would perform effectively with vector, pipeline or parallel computer architectures. It seemed well worthwhile to accept some limitations on the rate of convergence to a steady state in return for factors of 20 or more in processing speed which might be realized through the use of long vectors in a pipeline machine, or by the introduction of multiple parallel processors. Accordingly, we had concentrated on extracting the maximum possible efficiency from explicit time stepping schemes, which in principle would allow simultaneous processing of every point in the entire flow field. In the event, through the introduction of measures such as variable time steps based on the local stability limit, enthalpy damping [1], residual averaging [2], and

multigrid time stepping [3,4,5], it has proved possible to attain convergence sufficient for engineering predictions in about 50 cycles. Of these only the residual averaging would impose some limit on vectorization, but this restriction could be eliminated by substituting a Jacobi iterative scheme for the direct solution algorithm presently used. It has already been verified that this is effective in experiments with a scheme using a triangular mesh [6].

The present code is an adaptation of FLO57, a widely distributed code for three dimensional wing calculations. The initial multiblock scheme was programmed by Jameson during a visit to Dornier in the summer of 1982. It has subsequently been developed and refined principally by Leicher at Dornier, who also modified the scheme to allow the use of a multigrid solution algorithm in each block, and by Dawson at Cray Research. The original scheme was designed for disk storage. The consequent requirement of sequential access leads to some complications in the logical structure of the scheme which can be eliminated in situations where the data is held in a device allowing immediate access to arbitrarily located subsets. The solid state storage device of the CRAY X/MP, for example, allows blocks of data to be selected by a pointer. The same logical scheme for exchange of information across block boundaries of the grid could also be used for calculations on a multiprocessor machine with the different grid blocks assigned to different processors.

The multistage time stepping algorithm is briefly reviewed in the next section to provide a framework for a description of the logical structure of the multiblock scheme in Section 3. We discuss programming problems in Section 4, and finally we present an example of a calculation using 2.5 million mesh points in Section 5.

2. Description of the Algorithm

The algorithm is intended for use in steady state calculations of three dimensional compressible inviscid flows. It simulates the unsteady Euler equations, with modifications to accelerate convergence to a steady state. A semi-discrete approximation is first introduced by subdividing the domain into hexahedral cells. This reduces the equations to a set of ordinary differential equations, which are integrated by a multistage time stepping procedure. This separation of the space and time discretization procedures assures that the steady state is independent of the time stepping scheme.

We denote the pressure, density and three Cartesian velocity components as p, ρ, u, v and w. For a perfect gas we can express the total energy as

$$E = \frac{p}{(\gamma-1)\rho} + \frac{1}{2}(u^2 + v^2 + w^2) .$$

The total enthalpy is then defined by

$$H = E + p/\rho .$$

In integral form, the Euler equations can be written as

$$\frac{\partial}{\partial t}\iiint_{\Omega} w d\Omega + \iint_{\partial\Omega} \underline{F} \cdot d\underline{S} = 0 \qquad (1)$$

for a fixed region Ω with boundary $\partial\Omega$. w represents the conserved quantity and $\underline{F}$ is the corresponding flux across the boundary. The x momentum equation, for example, is obtained by setting

$$w = \rho u \quad \text{and} \quad \underline{F} = (\rho u^2 + p,\ \rho uv,\ \rho uw).$$

The dependent variables w are assumed to be known at the center (i,j,k) of each mesh cell. A semi-discretization leads to the equation

$$\frac{d}{dt}(V_{ijk}\, W_{ijk}) + Q(W)_{ijk} = 0 \tag{2}$$

Here Q_{ijk} represents the net flux out of the cell which is balanced by the rate of change of w in the cell whose volume is V. The flux is given by

$$Q_{ijk} = \sum_{\text{cell sides}} \underline{F} \cdot \underline{S}$$

where $\underline{F}$ is the flux at the center of a cell face and $\underline{S}$ denotes the cell face area. The value of $\underline{F}$ at the cell face is taken as the average of $\underline{F}$ at the cell centers on either side of the cell face.

In order to suppress the tendency for odd and even point decoupling and to capture shockwaves without any overshoots, it is necessary to add a dissipative term to equation (2). This leads to the semi-discrete equation

$$\frac{d}{dt}\,(V_{ijk}\, w_{ijk}) + Q_{ijk} - D_{ijk} = 0 \tag{3}$$

The term D_{ijk} is constructed so that it is of third order in smooth regions of flow. For the density equation $D_{ijk}(\rho)$ has the form

$$D_{ijk} = d_{i+1/2,j,k} - d_{i-1/2,j,k} + d_{i,j+1/2,k} - d_{i,j-1/2,k} + d_{i,j,k+1/2} - d_{i,j,k-1/2}$$

where

$$d_{i+1/2,j,k} = \frac{V_{i+1/2,j,k}}{\Delta t^*}\; \varepsilon^{(2)}_{i+1/2,j,k}\, \Delta_x\, \rho_{i,j,k} - \varepsilon^{(4)}_{i+1/2,j,k}\, \Delta_x^3\, \rho_{i-1,j,k}$$

and Δ_x is the forward difference operator defined by

$$\Delta_x \rho_{ijk} = \rho_{i+1,j,k} - \rho_{i,j,k}.$$

Also Δt^* is the time step corresponding to a nominal Courant number of unity, resulting in a normalization proportional to the maximum signal speed.

The coefficient $\varepsilon^{(2)}_{i+1/2,j,k}$ is made proportional to the normalized second difference of the pressure

$$\nu_{ijk} = \left| \frac{p_{i+1,j,k} - 2p_{ijk} + p_{i-1,j,k}}{p_{i+1,j,k} + 2p_{ijk} + p_{i-1,j,k}} \right|$$

in adjacent cells. This quantity is of second order except in regions containing a steep pressure gradient. The fourth differences provide background dissipation throughout the domain. In the neighborhood of a shockwave, ν_{ijk} is order one and the second differences become the dominant dissipative terms. The dissipative terms for the other equations are constructed from similar formulas with the exception of the energy equation, where the differences are of ρH rather than ρE. The purpose of this is to allow a steady state solution for which H remains constant.

The cell volume V_{ijk} is independent of time so we can write (3) as

$$\frac{dw}{dt} + R(w) = 0$$

where

$$R(w) = \frac{1}{V_{ijk}} (Q_{ijk} - D_{ijk})$$

The time integration is carried out by using a multistage scheme in conjunction with a multigrid strategy. If we let w^n and w^{n+1} represent the values of w at the beginning and end of the nth time step, then a k stage scheme for updating the fine grid values can be written as

$$w^{(0)} = w^n$$

$$w^{(q)} = w^{(0)} - \alpha_q \Delta t \; R^{(q-1,r)}, \quad q = 1,2,\ldots k$$

$$w^{n+1} = w^{(k)}$$

In the (q+1)st stage $R^{(q,r)}$ is evaluated as

$$R^{(q,r)} = \frac{1}{V_{ijk}}(Q_{ijk}(w^{(q)}) - D_{ijk}(w^{(s)}))$$

where $s = \min(q,r)$ with $0 \leq r < k$.

The coefficients $\{\alpha_q \mid q = 1,\ldots, k\}$ can be chosen to generate schemes with desirable stability properties. The parameter r determines the number of stages in which the dissipative terms are re-evaluated. Thus with r equal to zero, the dissipation is evaluated once and then frozen after the first stage; with r set equal to 1 the dissipation is evaluated twice. By freezing the dissipation prior to the final stage (typically r = 0 or 1) it is possible to tailor the multistage scheme to provide good damping of the amplification factor at high frequencies, and thus generate a scheme well suited to a multigrid strategy.

3. Logical Structure of the Multiblock Scheme

The program is designed to use a multiblock grid structure of the type illustrated in Figure 1. Here the grid blocks are arranged in a triply indexed array with block indices IB, JB, KB, while the grid points inside each block are represented by internal indices I, J, K. The data exchange problem could be simplified by a subdivision into planes, but the present subdivision is designed to allow maximum flexibility in the grid generation procedure. It appears likely that grid generation for a complex configuration can be simplified by a subdivision into subdomains which could be different blocks of the multiblock grid. Separate grid generation procedures can than be used within each block. The topological constraints imposed by the multiplicative structure is a very slight limitation. An L shaped region, for example, can be treated by filling out the array with empty sub-blocks. This procedure has been used by Fritz at Dornier in the generation of grids for the treatment of flows past cars.

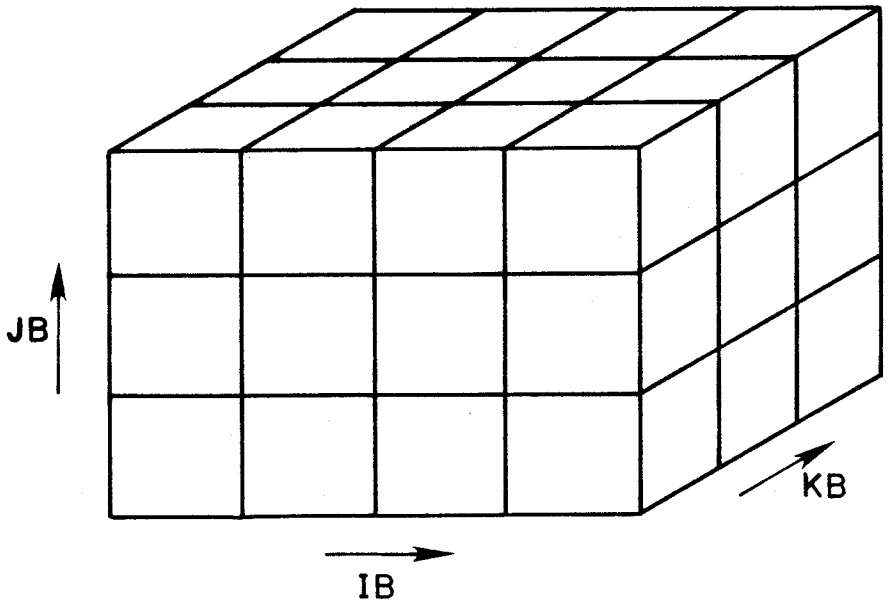

Figure 1

Multiblock grid with block indices IB, JB, KB.

The data exchange problem can be understood by referring to Figure 2, which illustrates a typical block. The grid indices inside the block range from I = 2 to IL, J = 2 to JL, K = 2 to KL. Cells external to the block are defined by index values I = 1, I2, J = 1, J2 and K = 1, K2. If the block boundary is a boundary of the whole flow field (either the surface of the configuration, or the outer boundary in the far field) then the values of the flow variables in the boundary cells must be determined by the boundary conditions. If the block boundary is an interface to a neighboring block, on the other hand, then the values in the boundary cells must be obtained from cells just inside the boundary of the next block.

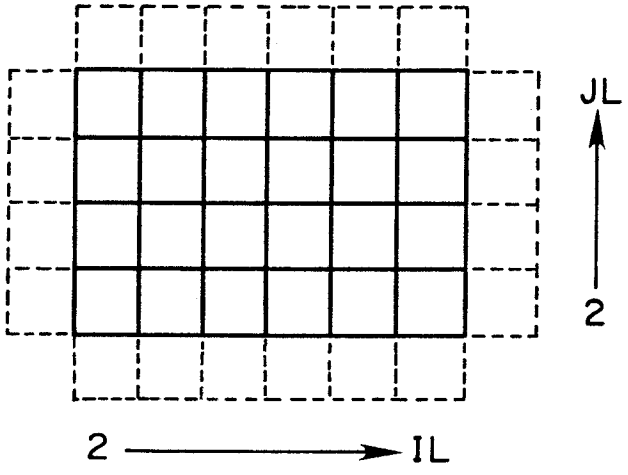

Figure 2

Data structure for a grid block in two dimensions.

Interior points range over I = 2,IL, J = 2,JL

In order to avoid the need to keep the neighboring blocks in core, which would require the storage of 7 blocks in the case of a fully interior block, we introduce buffers at all block boundaries. The following data structure serves this purpose. In the main database, which may be stored on a disk or a solid state storage device, we define a file MO to contain the values of the flow variables in all the blocks, and buffer files MI1, MI2, MJ1, MJ2, and MK1, MK2 to store the values at the boundaries of the blocks in the I, J and K directions. The flow variables inside a block are represented by the array W(I,J,K,N), where the index N ranges from 1 to 5 to represent the dependent variables (the density, three momentum components and the energy). To initialize W in a block for a stage of the multistage time stepping scheme we read W(I,J,K,N) for I = 2, 1L, J = 2,JL, K = 2,KL and N = 1,5 from MO. We read W(1,J,K,N) for J = 2,JL, K = 2,KL and N = 1,5 from MI1, and so on.

After updating the flow field inside the block the data is returned to the main database. In the case of disk storage, however, one would wish to read the data for the next block from the same files, and one cannot interleave read and write statements without returning to the beginning of the file. Moreover, the time stepping algorithm calls for the calculation of the correction in each cell from values in neighboring cells which have not yet been updated. Correspondingly each block should be treated using boundary values from neighboring blocks which have not yet been updated. Therefore the updated values are returned to a duplicate set of files NO, NI1, NI2, NJ1, NJ2 and NK1, NK2. The values just inside the left boundary provide the data for the right boundary of the block to the left. Thus we write W(2,J,K,N) for J = 2,JL, K = 2,KL, N = 1,5 into the file NI2. Correspondingly we write W(IL,J,K,N) for J = 2,JL, K = 2,KL,

N = 1,5 into the file NI1, and the buffers on the other boundaries are treated in the same way. Finally after all the blocks have been updated we interchange the names of the files, so that on the next stage the files which have just been written as N files are read as M files.

With random access to the storage device duplicate files would not be needed for the interior data in each block, but duplicate buffer files would still be needed at the block boundaries to preserve the data flow of the algorithm. With disk storage, however, the need to read the files in the same order in which they have been written leads to an additional complication. Boundary values at the outer boundary of the entire grid have to be determined from data coming from the same block. This means, for example, that a block along the bottom boundary must provide data for its own bottom boundary which would normally be placed in NJ1, and also the data for the block immediately above it. Referring to Figure 3, we can see that if the blocks were updated in the order indicated in Figure 3(a), which is also the order in which the MJ1 file would be read, then the NJ1 file would be written in the order indicated in Figure 3(b). In the existing code this incompatibility is eliminated by using separate buffer files MI0, MI3, MJ0, MJ3 and MK0, MK3 instead of the files MI1, MI2, MJ1, MJ2 and MK1, MK2 to contain values at all outer boundaries. The result of updating the interior buffer files as they are processed is then to place the data in the proper order for these files to be read.

9	10	11	12
5	6	7	8
1	2	3	4

(a)
Order in which blocks might be updated.

9	10	11	12
2	4	6	8
1	3	5	7

(b)
Corresponding order in which buffer file MJ1 would be updated.

Figure 3

A final complication is caused by the topology of the C mesh which is used in the code. Referring to Figure 4, it can be seen that this leads to the need to exchange data across the cut between blocks which are not contiguosly numbered. This requires the introduction of an additional set of buffer files for blocks separated by the cut.

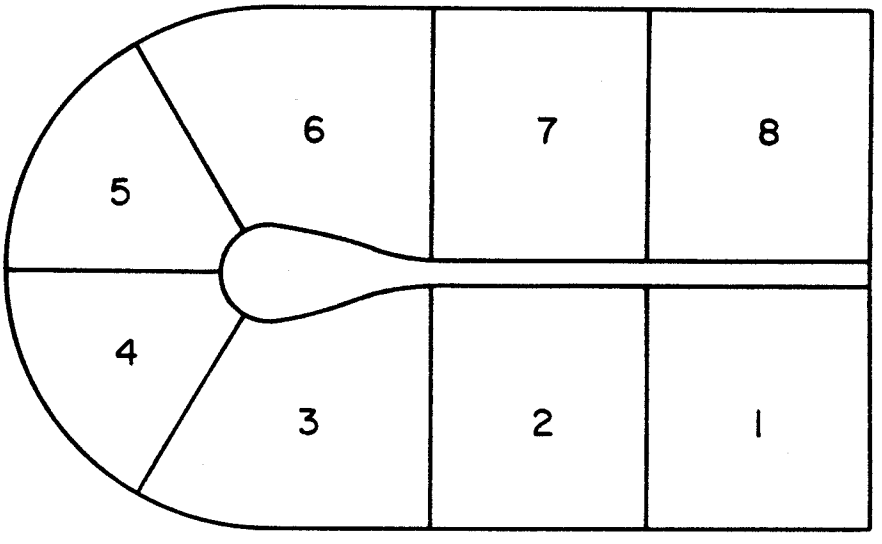

Figure 4

Interfaces introduced between non-contiguous blocks by C-mesh topology.

The implementation described here does not allow for the introduction of dissipative terms using fourth differences across boundaries, and in the existing code these higher order dissipative terms are simply switched off at the block boundaries. No difficulties have been encountered, but one could include fourth differences across boundaries without too much difficulty by using buffers containing 2 planes of data points instead of only a single plane.

The same logical structure for data exchange between blocks carries over to the situation in which the grid is divided into blocks which are simultaneously updated by separate processors. If the number of blocks equals the number of processors then the interior values might be retained in the separate memories of the different processors. If there are more blocks than processors one might still wish to transfer the interior values back and forth between local memories and a second level central memory. In any case the data exchange between blocks updated by different processors can be neatly accomplished by duplicate M and N boundary buffers in the same way as in the present code. It would then only be necessary to flag the completion of the updating process in all blocks before exchanging the names of the M and N files, and proceeding to the next sweep through the blocks.

4. Remarks on Programming

In carrying out the various data transfers it is important to avoid nested read or write statements. For example the transfer of data from the file MI1 at left hand block boundary might be accomplished by

```
READ (MI1)(((W(1,J,K,N),J=2,JL), K = 2,KL),N = 1,5)
```

This results in separate input/output operations for each array element. To avoid this we can define a buffer array BUFI(J,K,N) dimensioned to J2,K2,5.

Now one can write

```
READ (MI1) BUF1
```

followed by

```
      DO 10 N = 1, 5
      DO 10 K = 2, KL
      DO 10 J = 2, JL
      W(1,J,K,N) = BUFI(J,K,N)

   10 CONTINUE
```

The block dimensions may vary from one case to the next. To avoid wasteful transfers of meaningless data it is best to use variable dimensions for the array BUFI. This can be accomplished by placing the transfer statements inside separate subroutines. The efficiency of the data transfers is also improved by using the BUFFER IN and BUFFER OUT statements to allow asynchronous transfers which can be overlapped with the calculations.

With these measures the data transfer operations incur only a small penalty in processor time, even with disk storage. The use of a disk leads, however, to

a long residence time because of the time spent in data transfer during which the processor may be switched to the execution of other programs. Also, depending on the accounting system, there may be a substantial charge for the disk input/output operations themselves.

The volume of file transfer operations can be drastically reduced by modifying the algorithm so that all the stages of one time step are performed in each block before passing to the next block. Whereas the loop over the block indices IB, JB, KB would previously be inside the loop for the time stepping stages, it is now brought outside. The resulting restriction of the data flow between the blocks could have an adverse effect on the stability of the scheme. It has proved to work quite well in practice, however, incurring only a slight reduction in the rate of convergence to a steady state. One can go further and perform one complete multigrid cycle in each block before passing to the next block. This has been found to be quite effective in calculations performed by Leicher at Dornier. Experiments by Jong Yu at Boeing also indicate that in this situation, where any pretence of time accurate simulation has been abandoned, it pays to use the latest available values for the boundary data of all the blocks, so that an interior block would be treated with new values on some boundaries and old values on others, as in a Gauss-Seidel scheme. If the frequency of data exchanges between blocks is limited, it may also pay to use more complicated interface conditions taking account of the directions of wave propagation.

5. Example of a Multiblock Calculation with 2.5 Million Grid Points

The multiblock Euler code FLO57 is now in routine use at Dornier for prediction of flow about wings and wing body combinations. It was adapted this summer by Leicher and Dawson for use on the Cray X/MP with a solid state storage device (SSD) with a capacity of 128 million words. The results of a test calculation of the vortical flow past a delta wing are presented in Figure 5. A C-mesh was used, with 352 cells in the chordwise I direction, 64 cells in the normal J direction and 112 cells in the spanwise K direction giving a total of 2,523,136 volumes. The grid was subdivided in the J and K directions to produce 16 blocks each with 352x16x28 volumes. The configuration is the well known Dillner wing, at a Mach number of 7, and an angle of attack of 15 degrees. The figure shows pressure contours over the planform, and plots of the pressure coefficient across the span at two longitudinal stations, at 40% and 80% of the root chord. Preliminary calculations were performed using coarser meshes with 88x16x28 and 176x32x56 volumes, and the results on each mesh were used to provide initial values for the next mesh. A five stage time stepping scheme was used in which the dissipative terms were evaluated twice during each time step. This is the scheme which has proved most successful in multigrid calculations [4,5]. In this case residual averaging was used, allowing time steps corresponding to a Courant number of 6.5, but multigrid was not used. The complete calculation of 150 steps on the 88x16x28 mesh, followed by 150 steps on the 176x32x56 mesh, and 500 steps on the 352x64x112 mesh, took 7.9 hours of CPU time using a single processor of the Cray X/MP, and 8.3 hours of wall clock time. Thus the overhead incurred by the use of the SSD was about five percent.

The five stage algorithm requires 1422 floating point operations per mesh cell, including 368 for two evaluations of the dissipation, and 375 for the

residual averaging. The estimation of the permissable time step before each cycle requires another 74 operations, including 3 square roots, giving a total of 1496 operations per cell. With 2,523,136 cells the number of operations to advance one time step is thus about 3.77×10^9, not counting the additional operations needed to enforce the surface and farfield boundary conditions. Allowing for the preliminary calculations on the coarse and medium meshes in addition to 500 steps on the fine mesh, the total number of operations required for the treatment of the interior cells amounts to an aggregate for the three meshes of about 1.96×10^{12}. Thus the CPU time of 7.9 hours represents a sustained computing rate in the neighborhood of 70 megaflops. This performance is a consequence of the vectorization of every inner loop in the implementation of the entire algorithm.

6. Conclusion

These experiments clearly indicate the feasibility of using supercomputers of the current generation for the prediction of transonic flow past a complete aircraft by solution of the Euler equations for inviscid flow. Using a multi-block patched grid, it should be possible to resolve the main geometric features with a grid containing about a million cells, and with the present algorithm such a calculation would require a CPU time of the order of several hours. The main remaining difficulty lies in grid generation, and attrition of the accuracy and rate of convergence induced by grid irregularities and singularities. Research is needed on the development of discretization schemes which do not require grid regularity, and methods of obtaining rapid convergence to a steady state with a multi-block structure. Looking further ahead, the inclusion of viscous affects will be the next step. This could also be accomplished with computers of the current power by the introduction of turbulence models. The development of sufficiently reliable and accurate models is likely to be the pacing item of such a development. For a more detailed prediction based on large eddy simulation we must look forward to the appearance of machines such as the Cray 3 and ETA Systems GF10 in the vanguard of a new generation of computers.

References

1. Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time Stepping Schemes", AIAA Paper 81-1259, 1981.

2. Jameson, A., Baker, T.J., "Solution of the Euler Equations for Complex Configurations", Proc. AIAA 6th Computational Fluid Dynamics Conference, Danvers, 1983, pp. 293-302.

3. Jameson, A., "Solution of the Euler Equations by a Multigrid Method", Applied Mathematics and Computations", 13, 1983, pp. 327-356.

4. Jameson, A., and Baker, T.J., "Multigrid Solution of the Euler Equations for Aircraft Configurations", AIAA Paper 84-0093, 1984.

5. Baker, T.J., Jameson, A., Schmidt, W., "A Family of Fast and Robust Euler Codes", Proc. of CFD User's Workshop, University of Tennessee Space Institute, Tullahoma, 1984.

6. Jameson, A., and Mavripilis, D., "Finite Volume Solution of the Two-Dimensional Euler Equations on a Regular Triangular Mesh", AIAA Paper 85-0435, 1985.

PRESSURE COEFFICIENT

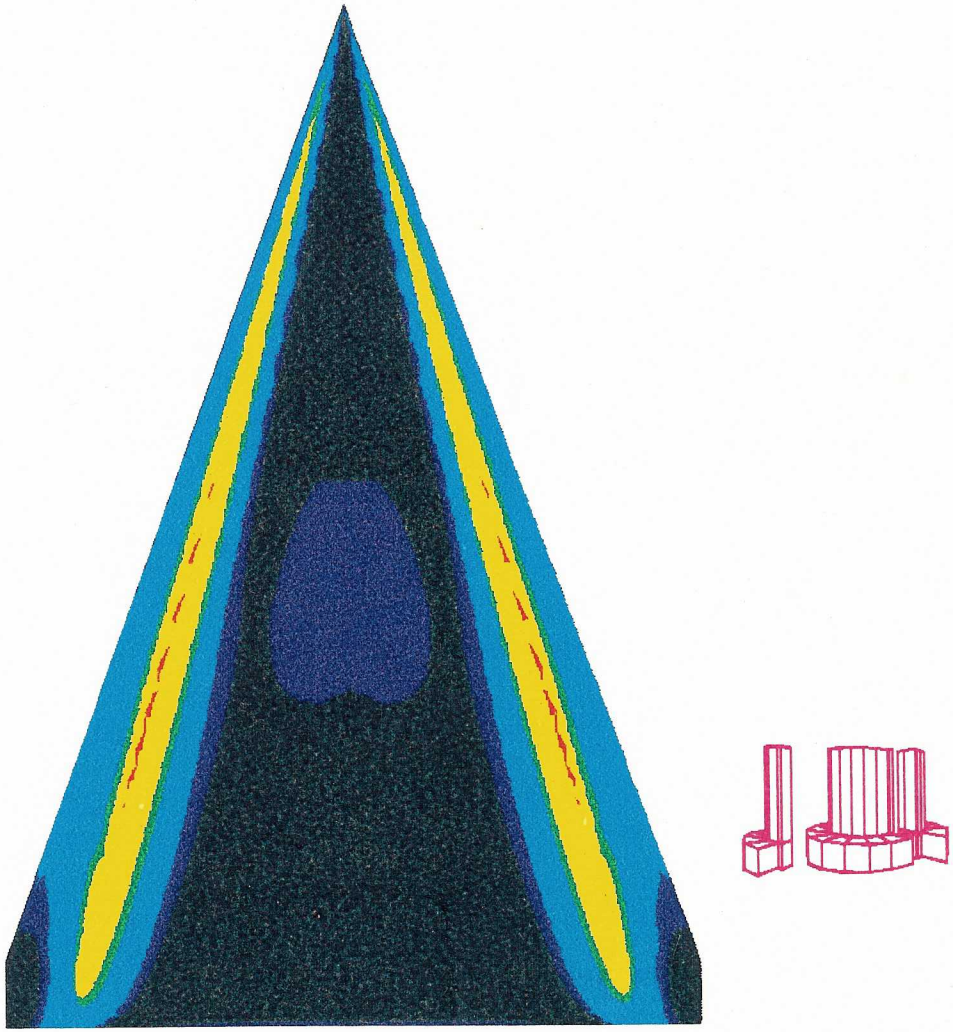

Figure 5

MACH NUMBER